Ejercicios

Unidad 8

Programación Orientada a Objetos

Estos ejercicios están pensados para empezar creando clases muy sencillas (apartado A) que luego irás mejorando y ampliando (apartados B, C...) de modo que practiques y aprendas poco a poco los aspectos fundamentales de la POO. Lo importante es que entiendas qué está pasando. Si no lo tienes claro "juega" con el código, haz pruebas "aver qué sucede si...", revisa la teoría, etc.

A. Clases simples con atributos

En cada ejercicio debes crear un programa con dos clases: una clase principal (puedes llamarla por ejemplo UD8_ProgramaPunto, según el ejercicio) que solo contendrá la función main, además de otra clase (con sus atributos y métodos) que utilizarás desde el main de la clase principal para hcer pruebas sobre su funcionamiento.

En este apartado las clases solo contendrán atributos (variables) y haremos algunas pruebas sencillas con ellas para entender cómo se instancia objetos y se accede a sus atirbutos. Por ahora no utilices ningún modificador en los atributos (public, private, static, final...).

A1. Punto

Crea un programa con una clase llamada Punto que representará un punto de dos dimensiones en un plano. Solo contendrá dos atributos enteros llamadas **x** e **y** (coordenadas).

En el main de la clase principal instancia 3 objetos Punto con las coordenadas (5,0), (10,10) y (-3, 7). Muestra por pantalla sus coordenadas (utiliza un println para cada punto). Modifica todas las coordenadas (prueba distintos operadores como = + - += *=...) y vuelve a imprimirlas por pantalla.

A2. Persona

Crea un programa con una clase llamada Persona que representará los datos principales de una persona: **dni**, **nombre**, **apellidos** y **edad**.

En el main de la clase principal instancia dos objetos de la clase Persona. Luego, pide por teclado los datos de ambas personas (guárdalos en los objetos). Por último, imprime dos mensajes por pantalla (uno por objeto) con un mensaje del estilo "Azucena Luján García con DNI ... es / no es mayor de edad".

A3. Rectángulo

Crea un programa con una clase llamada Rectangulo que representará un rectángulo mediante dos coordenadas (x1,y1) y (x2,y2) en un plano, por lo que la clase deberá tener cuatro atributos enteros: **x1**, **y1**, **x2**, **y2**.

En el main de la clase principal instancia 2 objetos Rectangulo en (0,0)(5,5) y (7,9)(2,3). Muestra por pantalla sus coordenadas, perímetros (suma de lados) y áreas (ancho x alto). Modifica todas las coordenadas como consideres y vuelve a imprimir coordenadas, perímetros y áreas.

A4. Artículo

Crea un programa con una clase llamada Articulo con los siguientes atributos: **nombre**, **precio** (sin IVA), **iva** (siempre será 21) y **cuantosQuedan** (representa cuantos quedan en el almacén).

En el main de la clase principal instancia un objeto de la clase artículo. Asígnale valores a todos sus atributos (los que quieras) y muestra por pantalla un mensaje del estilo "Pijama - Precio:10€ - IVA:21% - PVP:12,1€" (el PVP es el precio de venta al público, es decir, el precio con IVA). Luego, cambia el precio y vuelve a imprimir el mensaje.

B. Constructores

El constructor es el método que se ejecuta cuando se instancia un objeto. Si no está definido Java ejecutará un constructor por defecto que creará el objeto e inicializará todas sus variables a cero, pero esta no es una buena práctica. Es mejor definir nosotros un constructor que controle qué debe suceder cuando se cree el objeto.

En este apartado tienes que modificar los programas del apartado anterior (o haz una copia del proyecto si lo prefieres, o cópialos a otro paquete) y realizar los cambios indicados.

B1. Punto

Añade a la clase Punto un constructor con parámetros que copie las coordenadas pasadas como argumento a los atributos del objeto. Así:

```
public Punto(int x, int y)
{
    this.x = x;
    this.y = y;
}
```

Copiamos los valores pasados como argumento a los atirbutos del objeto. Ten en cuenta que int x e int y son variables locales del método, NO son los atributos del objeto. Para hacer referencia a los atributos del objeto hay que utilizar this.

Fíjate que ya no será posible hacer Punto p = new Punto(). Ahora será obligatorio hacer por ejemplo Punto p = new Punto(2, 7). En el apartado A tenías que recordar asignar valores a x e y tras crear un punto, lo cual no es una buena idea en proyectos grandes con cientos de objetos (es muy fácil equivocarse). Ahora es imposible equivocarse porque Java no te dejará. Hemos asegurado que todos los puntos siempre tendrán coordenadas.

Corrige el main y utiliza el constructor con parámetros para instanciar los objetos, pasándole como argumento los valores deseados.

B2. Persona

Añade a Persona el constructor de abajo y corrige el main para utilizarlo:

```
public Persona(String dni, String nombre, String apellidos, int edad) {
	this.dni = dni;
	this.nombre = nombre;
	this.apellidos = apellidos;
	this.edad = edad;
}
```

Ten en cuenta que no es obligatorio que los parámetros del constructor se llamen igual que los atributos del objeto (en tal caso no sería necesario utilizar this). Podríamos hacerlo así:

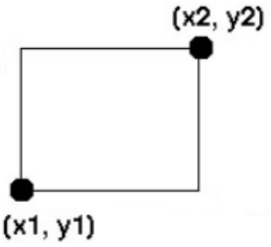
```
public Persona(String id, String nom, String ap, int e) {
	dni = id;
	nombre = nom;
	apellidos = ap;
	edad = e;
}
```

Tampoco es obligatorio pasar al constructor todos los atributos de la clase. Podríamos decidir por ejemplo que en nuestro software todas las personas deben tener nombre, apellidos y edad, pero no es obligatorio el DNI (recién nacidos y niños). Este constructor también sería válido:

```
public Persona(String nom, String ap, int e) {
	nombre = nom;
	apellidos = ap;
	edad = e;
}
```

Una clase puede tener tantos constructores como quieras siempre y cuando tengan distinto número y/o tipo de parámetros (para que no haya ambigüedad en cual utilizar).

B3. Rectángulo



En nuestro software necesitamos asegurarnos de que la coordenada (x1,y1) represente la esquina inferior izquierda y la (x2,y2) la superior derecha del rectangulo, como en el dibujo.

Añade a Rectangulo un constructor con los 4 parámetros. Incluye un if que compruebe los valores (*). Si son válidos guardará los parámetros en el objeto.

Si no lo son mostrará un mensaje del estilo “ERROR al instanciar Rectangulo...” utilizando System.err.println(...). No podremos evitar que se instancie el objeto pero al menos avisaremos por pantalla.

Corrige el main para utilizar dicho constructor. Debería mostrar un mensaje de error.

(*) Pista: Es suficiente con un if ((condición) && (condición))

B4. Artículo

Añade un constructor con 4 parámetros que asigne valores a nombre, precio, iva y cuantosQuedan. Dicho constructor deberá mostrar un mensaje de error si alguno de los valores nombre, precio, iva o cuantosQuedan no son válidos. ¿Qué condiciones crees que podrían determinar si son válidos o no? Razónalo e implementa el código.

Corrige el main y prueba a crear varios artículos. Introduce algunos con valores incorrectos para comprobar si avisa del error.

C. Getters y Setters

Un pilar fundamental de la programación orientada a objetos (POO) es el encapsulamiento:

> "Se denomina encapsulamiento al ocultamiento del estado, es decir, de los datos miembro de un objeto de manera que solo se pueda cambiar mediante las operaciones definidas para ese objeto.
>
> Cada objeto está aislado del exterior. El aislamiento protege a los datos asociados a un objeto contra su modificación por quien no tenga derecho a acceder a ellos, eliminando efectos secundarios e interacciones. De esta forma el usuario de la clase puede obviar la implementación de los métodos y propiedades para concentrarse solo en cómo usarlos.
>
> Por otro lado se evita que el usuario pueda cambiar su estado de maneras imprevistas e incontroladas." Fuente: Wikipedia

Por ello, una práctica muy habitual en POO consiste en ocultar todos los atributos (hacerlos private) para que no se puedan modificar directamente desde fuera de la clase. En su lugar añadiremos métodos getters (get = coger) y setters (set = fijar) visibles (public) que permitan leer y modificar dichos atributos desde fuera de la clase. La clave está en que al tratarse de métodos podremos incluir el código necesario para **controlar el acceso a los atributos y protegerlas de usos incorrectos**.

En este apartado tienes que modificar los programas del apartado anterior (o haz una copia del proyecto si lo prefieres) y **realizar los cambios indicados.**

C1.Punto

Modifica los atributos de Punto para que sean **private**. Fíjate que desde el main ya no te dejará utilizar ni modificar los atributos x e y de los objetos.

Vamos a añadir los **getters**: int getX() e int getY() que devolverán los valores de x e y respectivamente. Es una forma indirecta de leer sus valores.

Añadiremos también los **setters**: void setX(int x) y void setY(int y) que copiarán el valor pasado como parámetro a los atributos de la clase.

Tanto getters como setters deben ser **public**.

Corrige el main para utilizar los getters y setters. Prueba a instanciar varios objetos, mostrar sus valores por pantalla, modificarlos, etc.

```
public class Punto {

    private int x;
    private int y;

    public Punto(int x, int y) {
        this.x = x;
        this.y = y;
    }

    public int getX(){
        return x;
    }

    public int getY(){
        return y;
    }

    public void setX(int x) {
        this.x = x;
    }

    public void setY(int y) {
        this.x = y;
    }
}
```

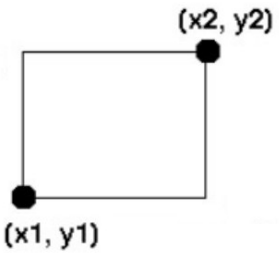
C2.Persona

Aplica el encapsulamiento básico a la clase Persona: Declara todos sus atributos como private y crea todos los getters y setters necesarios (un get y un set por atributo).

Corrige el main para utilizar los getters y setters. Prueba a instanciar varios objetos, mostrar sus valores por pantalla, modificarlos, etc.

C3.Rectángulo

Aplica el encapsulamiento básico a la clase Rectángulo: Declara todos sus atributos como private y crea todos los getters y setters necesarios (un get y un set por atributo).



¿Recuerdas la condición explicada en B3? Tendrás que programar los setters de modo que comprueben el valor pasado como argumento antes de guardarlo en el objeto. Si no fuera correcto se mostrará un mensaje de error (y NO se guardará el valor).

Corrige el main para utilizar los getters y setters. Prueba a instanciar varios objetos, mostrar sus valores, modificarlos, etc. Prueba varios valores erróneos para comprobar si funciona.

C4.Artículo

Aplica el encapsulamiento básico a la clase Articulo: Declara todos sus atributos como private y crea todos los getters y setters necesarios (un get y un set por atributo).

Programa los setters para que comprueben los valores y los guarden en el objeto solo si son correctos. En caso contrario muestra un mensaje de error.

D. Añadiendo métodos útiles

Una clase bien diseñada debería incluir métodos que realicen operaciones con la información de los objetos. De ese modo la clase dispondrá de funcionalidades útiles tanto para nosotros como para otros programadores. Esto es una práctica habitual y muy recomendable.

Por ejemplo, la clase Scanner tiene métodos como getInt(), getDouble, getLine(), etc. que alguien ha programado y podremos utilizar cuando los necesitemos sin tener que programarlo nosotros ni preocuparnos por cómo funcionan internamente. Lo mismo sucede con la clase String (charAt, substring, toCharArray, etc.), la clase Math (random, min, max, abs, etc.), la clase Arrays, etc.

Todas estas son clases que Java incorpora por defecto (hay miles). Existen muchas otras que permiten hacer todo tipo de cosas como crear interfaces gráficas con botones, trabajar con imágenes, leer y escribir en archivos, reproducir música, enviar o recibir datos a través de la red, etc.

En esta unidad estamos aprendiendo a diseñar y programar nuestras propias clases, los fundamentos de la Programación Orientada a Objetos, necesario en cualquier proyecto software de cierta envergadura.

Recuerda que los métodos pueden ser **private** (solo pueden utilizarse desde dentro de la clase) o **public** (pueden utilizarse desde fuera, forman parte de la interfaz de la clase).

En este apartado tienes que modificar los programas del apartado anterior (o haz una copia del proyecto si lo prefieres) y **añade los métodos que se indican**.

D1. Punto

Añade a la clase Punto los siguientes métodos públicos:

- **public void imprime()** // Imprime por pantalla las coordenadas. Ejemplo: “(7, -5)”
- **public void setXY(int x, int y)** // Modifica ambas coordenadas. Es como un setter doble.
- **public void desplaza(int dx, int dy)** // Desplaza el punto la cantidad (dx,dy) indicada. Ejemplo: Si el punto (1,1) se desplaza (2,5) entonces estará en (3,6).
- **public int distancia(Punto p)** // Calcula y devuelve la distancia entre el propio objeto (this) y otro objeto (Punto p) que se pasa como parámetro: distancia entre dos coordenadas.

Prueba a utilizar estos métodos desde el main para comprobar su funcionamiento.

D2. Persona

Añade a la clase Persona los siguientes métodos públicos:

- **public void imprime()** // Imprime la información del objeto: "DNI:... Nombre:... etc."
- **public boolean esMayorEdad()** // Devuelve true si es mayor de edad (false si no).
- **public boolean esJubilado()** // Devuelve true si tiene 65 años o más (false si no).
- **public int diferenciaEdad(Persona p)** // Devuelve la diferencia de edad entre 'this' y p.

Prueba a utilizar estos métodos desde el main para comprobar su funcionamiento.

D3. Rectángulo

Añade a la clase Rectángulo métodos públicos con las siguientes funcionalidades:

- Método para **imprimir** la información del rectángulo por pantalla.
- Métodos setters dobles y cuadruples: **setX1Y1**, **set X2Y2** y **setAll**(...).
- Métodos **getPerimetro** y **getArea** que calculen y devuelvan el perímetro y área del objeto.

Prueba a utilizar estos métodos desde el main para comprobar su funcionamiento.

D4. Artículo

Añade a la clase Artículo métodos públicos con las siguientes funcionalidades:

- Método para **imprimir** la información del artículo por pantalla.
- Método **getPVP** que devuelva el precio de venta al público (PVP) con iva incluido.
- Método **getPVPDescuento** que devuelva el PVP con un descuento pasado como argumento.
- Método **vender** que actualiza los atributos del objeto tras vender una cantidad 'x' (si es posible). Devolverá true si ha sido posible (false en caso contrario).
- Método **almacenar** que actualiza los atributos del objeto tras almacenar una cantidad 'x' (si es posible). Devolverá true si ha sido posible (false en caso contrario).

E. Static y Final

Los modificadores static y final son opcionales, pueden utilizarse tanto en atributos como en métodos y pueden combinarse ambos:

- **static**: El atributo o método pertenece a la clase (no al objeto). Por ello puede utilizarse sin instanciar ningún objeto, desde el nombre de la clase: NombreClase.atributo o NombreClase.metodo(...). Como el valor se almacena en la clase NO toma valores distintos en cada objeto (como sí sucede con los atirbutos 'normales' no static). Por ejemplo:
  - El atributo salarioMinimo de Empleado (ss común para todos, puede cambiar).
  - Métodos útiles como Arrays.fill(...) o String.valueOf(...) que podemos utilizar directamente desde las clases Arrays y String sin instanciar un objeto.

  Es importante saber que desde un método static no se pueden utilizar atributos ni métodos no static. Al contrario sí es posible.
- **final**: Un atributo final no se puede modificar. Puede tener valores distintos en cada objeto, pero debe fijarse su valor en el constructor. Un método final no se puede redefinir en una sub-clase heredada (veremos herencia en la siguiente unidad). Por ejemplo:
  - El DNI de la clase Persona. No puede cambiar y cada objeto tiene el suyo.
- La combinación de static y final combina ambas características. Por ejemplo:
  - El atributo Math.PI es static y final. Pertenece a la clase y no puede cambiar.

E1. Punto

Necesitamos un método que nos permita crear un objeto Punto con coordenadas aleatorias. Esta funcionalidad no depende de ningún objeto concreto por lo que será estática. Deberá crear un nuevo Punto (utiliza el constructor) con x e y entre -100 y 100, y luego devolverlo (con return).

- public static Punto creaPuntoAleatorio()

Pruebalo en el main para comprobar que funciona. Crea varios puntos aleatorios con Punto.creaPuntoAleatorio() e imprime su valor por pantalla.

E2. Persona

El DNI de una persona no puede variar. Añade el modificador final al atributo dni y asegúrate de que se guarde su valor en el constructor. Quita el método setDNI(...) que de todos modos ya no se podrá utilizar porque Java no te dejará modificar el atributo dni.

La mayoría de edad a las 18 años es un valor común a todas las personas y no puede variar. Crea un nuevo atributo llamado mayoriaEdad que sea static y final. Tendrás que inicializarlo a 18 en la declaración. Utilízalo en el método que comprueba si una persona es mayor de edad.

Crea un método **static boolean validarDNI(String dni)** que devuelva true si dni es válido (tiene 8 números y una letra). Si no, devolverá false. Utilízalo en el constructor para comprobar el dni (si no es válido, muestra un mensaje de error y no guardes los valores).

Realiza algunas pruebas en el main para comprobar el funcionamiento de los cambios realizados. También puedes utilizar Persona.validarDNI(...) por ejemplo para comprobar si unos DNI introducidos por teclado son válidos o no (sin necesidad de crear ningún objeto).

E3. Rectángulo

Necesitamos hacer algunos cambios para que todas las coordenadas estén entre (0,0) y (100,100). Añade a la clase Rectángulo dos atributos llamados min y max. Estos valores son comunes a todos los objetos y no pueden variar. Piensa qué modificados necesitas añadir a min y max.

Utiliza min y max en el constructor y en los setters para comprobar los valores (como de costumbre, si no son correctos muestra un mensaje de error y apliques los cambios).

También necesitamos un método no constructor para crear rectángulos aleatorios. Impleméntalo.

Realiza pruebas en el main para comprobar su funcionamiento.

E4. Artículo

En España existen tres tipos de IVA según el tipo de producto:

- El IVA general (21%): para la mayoría de productos a la venta.
- El IVA reducido (10%): hostelería, transporte, vivienda, etc.
- El IVA super reducido (4%): alimentos básicos, libros, medicamentos, etc.

Estos tres tipos de IVA no pueden variar y a cada artículo se le aplicará uno de los tres.

Razona qué cambios sería necesario realizar a la clase Articulo e impleméntalos.

Segunda parte

1. Clase Triángulo

- Crea una clase Triangulo. Sus atributos son los 3 puntos de sus vértices
- Crea un constructor al que se le pasan los tres vértices (recuerda que son Puntos)
- Crea un método que devuelva la longitud del perímetro del triángulo.

2. Consumo eléctrico

Escriba un programa para representar el consumo de energía de una instalación eléctrica. Para ello, se dispondrá de una clase que representa los aparatos conectados en la instalación. Cada aparato tiene un consumo eléctrico determinado. Al encender un aparato el consumo eléctrico se incrementa en la potencia de dicho aparato. Al apagarlo, se decrementa el consumo. Inicialmente los aparatos están todos apagados.

Hacer un programa que declare tres aparatos electricos, una bombilla de 100 watios, un radiador de 2000 watios y una plancha de 1200 watios. El programa imprimirá el consumo nada más crear los objetos. Posteriormente, se enciende la bombilla y la plancha, y el programa imprime el consumo. Luego se apaga la plancha y se enciende el radiador y se vuelve imprimir el consumo.

3. Bombillas

Se desea representar las bombillas que pueda haber en una casa. Cada bombilla tiene asociada un interruptor y sólo uno. Así mismo, existe un interruptor general de la casa que en un principio está apagado. Un interruptor tiene dos estados, ON y OFF. Una bombilla luce si el interruptor general de la casa está en ON y su interruptor asociado también.
Escriba una clase de nombre Bombilla que permita modelar la información anterior. Para ello la clase dispondrá de :

- Un método para cambiar el estado del interruptor de al bombilla.
- Un método para cambiar el estado del interruptor general de la casa.
- Un método que determina si una bombilla esta luciendo o no.

Para probarlo, crea un programa que cree una bombilla y se imprima por pantalla si luce o no, luego se pulsa el interruptor de la bombilla y vuelve a imprimir es estado de la misma. Por último se pulsa en interruptor general y se vuelve a imprimir el estado de la bombilla.

4. EjemplarLibro

Implementar una clase llamada EjemplarLibro que se va a usar en una biblioteca y que tiene los siguientes atributos:

- Título de libro (se le pone en el momento del alta)
- Fecha de adquisición (es la fecha del sistema en el momento del alta) (Usa LocalDate.now() para obtener la fecha del sistema como cadena de texto, nos devuelve un objeto de tipo LocalDate mas información sobre fechas en java)
- Número de ejemplar: 1, 2, 3, etc. (de un mismo libro, la biblioteca tiene varios ejemplares)
- Prestado (sí /no).

Inicialmente no está prestado.Además, tiene los siguientes métodos:

- Constructor 1: cuando es el primer ejemplar de un determinado título, se le pasa como parámetro solo el título del libro. El resto de datos los puede calcular él.
- Constructor 2: se le pasa como parámetro un libro y copia todos sus atributos salvo el número de ejemplar que será 1 más el del libro pasado.
- Prestar(): si no está prestado lo marca como prestado y devuelve true¸si está prestado no hace nada y devuelve false.
- Devolver(): si está prestado lo marca como no prestado y devuelve true¸si no está prestado no hace nada y devuelve false.
- toString(): Genera un String con el nombre, la fecha entre paréntesis y el número de ejemplar entre corchetes. Este método se usará para sacar por pantalla de forma cómoda los datos de un libro.

Haz un main() que cree 4 libros (probando ambos constructores), que realice algún préstamo y devolución, y finalmente muestre los libros -con toString()-.

5. Receta

Implementa las clases Ingrediente y Receta (partiendo de los modelos UML que están anexos abajo). Una vez creadas las clases. Implementa un método main que introduzca por teclado los valores de los atributos para crear una receta. Se preguntará cuántos ingredientes lleva la receta. Una vez introducidos los valores mostrar la receta.

Receta
-Nombre(String) -Elaboracion(String) -Duración(int) -ingredientes (Ingrediente [])
+Receta(String, String, int, Ingrediente[]) +setNombre(String):void +getNombre():String +setElaboracion(String):void +getElaboracion():String +setDuracion(int):void +getDuracion():int +mostrarReceta():void

Ingrediente
-Nombre(String) -Cantidad (int) -Unidad(String)
+Ingrediente(String, float, String,) +setNombre(String):void +getNombre():String +setCantidad(int):void +getCantidad():int +setUnidad(String):void +getUnidad():String

6. Trabajador

Crea una clase Trabajador que tenga los siguientes atributos encapsulados:

- Importe de la hora extra (double), será un atributo de clase
- Número de trabajadores (int), será un atributo de clase
- DNI (String)
- Nombre (String)
- Sueldo base (double)
- Horas extras realizadas en el mes (int)
- Tipo de IRPF (%) (double)

Los objetos Trabajador se podrán crear con el constructor por defecto o con un constructor conteniendo el dni, nombre, sueldo base, horas extras realizadas en el mes y tipo de irpf.

Además de los métodos getter y setter de los atributos, tendrá los siguientes métodos:

- Método para el cálculo del complemento correspondiente a las horas extra realizadas calcularImporteHorasExtras().
- Método para calcular el sueldo bruto (sueldo base + complemento por horas extras) calcularSueldoBruto()
- Método para calcular las retenciones por IRPF. El porcentaje de IRPF se aplica sobre el sueldo bruto. calcularRetencionIrpf()
- Método para calcular el sueldo (sueldo bruto - retenciones) calcularSueldo().
- Método toString() para mostrar los datos de los trabajadores de la siguiente forma:

```
12345678A Pepito Grillo
Sueldo Base: 1150.0
Horas Extras: 4
tipo IRPF: 15.0
Sueldo Bruto: 1230.0
Retencion por IRPF: 184.5
Sueldo Neto: 1045.5
```

- Método leerTrabajador() que permite introducir los datos de un trabajador por teclado. A este método se le llama desde el main mediante una instancia de la clase Trabajador

```
Nombre: Pepito Grillo
DNI: 12345678A
Sueldo Base: 1150
Horas Extras: 4
tipo IRPF: 15
```

Una vez creada la clase Trabajador, la utilizaremos en un programa Principal que contenga el main y que realice las siguientes operaciones:

- Introducir por teclado el precio de las horas extraordinarias, común a todos los trabajadores.
- Crear un objeto t1 de la clase Trabajador y pedir los datos llamando al método leerTrabajador de la clase Trabajador.
- Crear un objeto t2 de la clase Trabajador y pedir los datos llamando a un método estático que tenemos que crear en la clase Principal y que se llama leerTrabajador al que se le pasa el objeto trabajador (utilizar mismo formato de petición de datos).
- Siguiendo en el main, pedir por teclado los datos de un nuevo trabajador utilizando el mismo formato de introducción de datos y construyendo un objeto t3 con el segundo constructor.
- Imprimir el número de trabajadores.
- Imprimir las nóminas de los tres trabajadores introducidos.

7. MovilPrepago

Queremos crear una clase para representar los móviles prepago (los jovencitos que no sepan lo que son, que pregunten al señor mayor del profesor).
Las propiedades de un móvil prepago son:

- int numeroMovil (13 digitos)
- float costeEstableceLlamada (euros, con dos decimales)
- float costeMinutoLlamada (euros, con dos decimales)
- float costeConsumoMB (euros, con dos decimales)
- float saldo (euros, con dos decimales) nunca puede ser negativo

Y los métodos que queremos implementar son:

- void efectuarLlamada (int segundos): reduce el saldo. Si el saldo no es suficiente, se corta la llamada
- void navegar(int MB) análogo a efectuar una llamada
- boolean recargar (int importe): aumenta el saldo, debe ser múltiplo de 5 euros, sino devuelve false
- float consultarSaldo ()

Al constructor de la clase se le deben pasar todos los datos necesarios para un nuevo móvil prepago.

Crea una clase para probar la clase MovilPrepago, crea un par de móviles, efectúa llamadas, navega, recarga y muestra cómo va modificándose el saldo.

8. EticalBank

La empresa LibreCoders te ha contratado para desarrollar un software de gestión de una cuenta bancaria para la cooperativa de banca ética y sostenible EticalBank. Se trata de una aplicación Java formada por una clase principal EticalBank y otra llamada CuentaBancaria.

El programa pedirá los datos necesarios para crear una cuenta bancaria. Si son válidos, creará la cuenta y mostrará el menú principal para permitir actuar sobre la cuenta. Tras cada acción se volverá a mostrar el menú.

1. Datos de la cuenta. Mostrará el IBAN, el titular y el saldo.

2. IBAN. Mostrará el IBAN.
3. Titular. Mostrará el titular.
4. Saldo. Mostrará el saldo disponible.
5. Ingreso. Pedirá la cantidad a ingresar y realizará el ingreso si es posible.
6. Retirada. Pedirá la cantidad a retirar y realizará la retirada si es posible.
7. Salir. Termina el programa.

Clase CuentaBancaria

Una cuenta bancaria tiene como datos asociados el iban (international bank acount number, formado por dos letras y 22 números, por ejemplo ES6621000418401234567891), el titular (un nombre completo) y el saldo (dinero en euros)

Cuando se crea una cuenta es obligatorio que tenga un iban y un titular (que no podrán cambiar nunca). El saldo será de 0 euros y la cuenta no tendrá movimientos asociados.

El saldo solo puede variar cuando se produce un ingreso (entra dinero en la cuenta) o una retirada (sale dinero de la cuenta). Los ingresos y retiradas solo pueden ser de valores superiores a cero.

El saldo de una cuenta nunca podrá ser inferior a -50(*) euros. Si se produce un movimiento que deje la cuenta con un saldo negativo (no inferior a -50) habrá que mostrar el mensaje “AVISO: Saldo negativo”. Si se produce un movimiento superior a 3.000(*) euros se mostrará el mensaje “AVISO: Notificar a hacienda”.

No se realizará ningún tipo de entrada por teclado. La única salida por pantalla permitida son los dos mensajes de aviso mencionados arriba, ninguna otra.

(*) Estos valores no pueden variar y son iguales para todas las cuentas bancarias.

Clase DawBank

Clase principal con función main. Encargada de interactuar con el usuario, mostrar el menú principal, dar feedback y/o mensajes de error, etc. Utilizará la clase CuentaBancaria. Puedes implementar las funciones que consideres oportunas.

9. MasterMind (opcional)

Haz el juego del “MasterMind” de la siguiente forma: la máquina genera al azar 4 dígitos entre 0 y 10 sin repetidos que es lo que se tendrá que adivinar. El jugador introducirá combinaciones de 4 dígitos también sin repetidos hasta que lo adivine. Cada vez que introduce una combinación el sistema le dirá cuántos dígitos de los introducidos están en la combinación a descubrir distinguiendo si están en la misma posición o si están en la combinación, pero en otra posición. Tiene 10 intentos para averiguarlo.

Ejemplo: combinación a adivinar: 9871 intento: 8471: 2 dígitos bien colocados, 1 dígito mal colocado

El sistema validará que las combinaciones introducidas sean de 4 posiciones y sin repetidos. Haz una clase con la lógica del juego y un programa que la utilice pero solo como intermediario entre la clase y el jugador. Trata también de hacer el ejercicio lo suficientemente flexible para que a futuro se pudiese fácilmente cambiar a combinaciones de 5 dígitos (ó 6 ó 7 etc.) y también cambiar el número de intentos.